GGCHEMPY: a Gas-Grain CHEMical code in pure PYthon

Jixing Ge
Email: gejixing666@gmail.com, gejixing666@163.com

August 9, 2021

Contents

1 Introduction

GGCHEMPY is a pure Python-based version of gas-grain chemical model for chemistry in interstellar clouds developed on basis of the GGCHEM code written in Fortran which have been used for various chemical problems (Ge, He & Yan 2016, Ge, He & Li 2016, Tang et al. 2019, Ge, Mardones, He, Rawlings, Liu, Lee, Tatematsu, Liu, Zhu, Chang, Inostroza & Feng 2020, Ge, Mardones, Inostroza & Peng 2020). It is developed to reach comparable speed to the versions written in Fortran but with only Python and necessary Python packages. In this way, it can be easily used to handle increasing reactions from new experiments and to combine with radiation transfer (RT) analysis as that have been done by e.g. Tafalla et al. (2004), Tafalla et al. (2006), Tritsis et al. (2018) and Sipilä et al. (2019). A Graphical user interface (GUI) is also provided using the PyQt5 package.

2 The GGCHEMPY code

2.1 Code frame

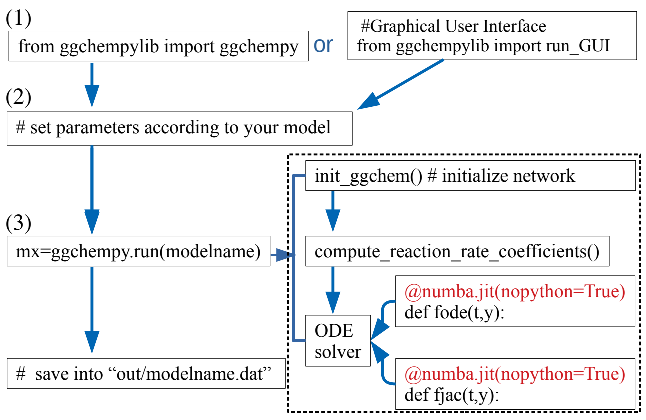

(1)
from ggchempylib import ggchempy
or
#Graphical User Interface
from ggchempylib import run_GUI
(2)
# set parameters according to your model
(3)
mx=ggchempy.run(modelname)
# save into "out/modelname.dat"
init_ggchem() # initialize network
compute_reaction_rate_coefficients()
ODE solver
@numba.jit(nopython=True)
def fode(t,y):
@numba.jit(nopython=True)
def fjac(t,y):

Figure 1: Frame of GGCHEM and numba.jit-accelerated parts (red color).

Table 1: Variables and default values in GGCHEMPY code (in `ggchempylib.ggchempy`).

Variable	Default value	Meaning
Gas parameters:		
`gas.nH`	2×10^4 (cm^{-3})	Gas density
`gas.T`	10 (K)	Gas temperature
`gas.Av`	10 (mag)	Visual extinction
`gas.Zeta`	1.3×10^{-17} (s^{-1})	Cosmic-ray ionization rate
`gas.Chi`	1.0 (χ_0)	Radiation field in unit of χ_0 (Draine 1978)
Dust parameters:		
`dust.nd`		Calculated using `gas.nH` and `ggpars.d2gmr`
`dust.T`	10 (K)	Dust temperature
`dust.radius`	10^{-5} (cm)	Dust radius
`dust.rho`	3.0 (g cm^{-3})	Dust mass density
`dust.surface.Rdb`	0.5	Diffusion-to-binding energy ratio
`dust.site_density`	1.5×10^{15} (site cm^{-2})	Site density
`dust.fc`	3.0×10^{-19}	Timescale ratio between dust cooling via desorption of molecules and subsequent heating events
`dust.Tpeak`	70.0 K	Peak temperature of dust grains heated by cosmic ray particle
Common parameters:		
`ggpars.d2gmr`	0.01	Dust-to-gas mass ratio
`ggpars.nstr`	8	String length for each species
`ggpars.nrnp`	8	Total number of reactants (3) and products (5)
`ggpars.ti`	1 (yr)	Initial time
`ggpars.tf`	10^9 (yr)	Final time
`ggpars.nt`	100	Number of time step in logarithmic-space
`ggpars.yr_sec`	3.1536×10^7 (s)	Year in unit of second
`ggpars.rtol`	1.0×10^{-05}	Relative tolerance
`ggpars.atol`	1.0×10^{-15}	Absolute tolerance
`ggpars.ggfiles[0]`	'in/network2.txt'	File of reaction network
`ggpars.ggfiles[1]`	'in/ed.txt'	File of binding energies
`ggpars.ggfiles[2]`	'in/iabun.txt'	File of initial abundances.
`ggpars.outdir`	out/'	Output directory.
Switches (integer):		
`iswitch.iNTD`	0	Non-thermal desorption. 0: OFF; 1 = Garrod et al. (2007); 2 = Minissale et al. (2016)
`iswitch.iSS`	0	Self-shielding of H_2 and CO. 0:OFF; 1:ON.

There are six classes in `ggchempylib.ggchempy`: `gas, dust, ggpars, elements, species, reactions`. Thus, physical and chemical properties can be defined separately in the five former classes to make study more clear.

GGCHEMPY is designed straightly as the work flow. The frame of GGCHEMPY is shown in Fig. 1. Three necessary steps are needed:

(1) Import `ggchempy` from `ggchempylib` or use a graphical user interface (GUI) e.g. `from ggchempylib import ggchempy, run_GUI`.

(2) Set parameters according to your model. If no parameters are modified, the defaults ones for dark clouds are used (see Table 1).

(3) Run the code by typing `ggchempy.run(file)` or by pressing the button '`run`' on the GUI. GGCHEMPY will call `ggchempy.init_ggchem()` to initialize reaction network according to your input parameters. Then GGCHEMPY computes reaction rate coefficient for all reactions by calling `ggchempy.compute_reaction_rate_coefficients()`. Finally, to solve the ODEs, BDF method from `scipy.integrate.ode` is used in the integration part. Here, `numba.jit(nopython=True)` is simply combined to Python functions `fode(y,t)` and `fjac(y,t)` to accelerate the integration of ODEs, which need massive calculations and loops (see red text in Fig. 1). The modeled results will be saved in a file with path and name of `"out/modelname.dat"` when `ggpars.outdir="out/"`. The modeled can also be directly passed to a Python dictionary, such as `mx` in Fig. 1, for analysis in the same Python script.

2.2 Format of input files

For the reaction network in "in/network2.txt", the number of elements, species and reactions should be given in the first line. Then three blocks of elements, species and reaction list are listed and separated by a blank line. GGCHEMPY does not need fixed format of element and species list. However, for the reaction list, the default format used in the reaction network of Semenov et al. (2010) is required: "8A8,3E9.2,I2" (in Fortran).

For the binding energies of species in "in/ed.txt", the number of species with binding energies should be given in the first line. Then a blank line is needed. Finally, the list of species and their binding energies should be given. No fixed format is needed for the list. GGCHEMPY will read the list and match the surface species to have the corresponding binding energies.

For the initial abundances of species in "iabun.txt", the number of species with initial abundances should be given in the first line. Then the list of species and initial abundance should be given.

2.3 Usage

To install GGCEHMPY, type the following command on your terminal:

```
python setup.py build
python setup.py install
```

Thus, you can use GGCHEMPY anywhere on your system.

To run a model with default parameters for typical dark clouds (see the default parameters in Table 1), after importing ggchemlib, ones can directly run GGCHEMPY using the following code:

```
from ggchempylib import ggchempy
DC=ggchempy.run('DC') # see model result in file of "out/DC.dat"
```

The simulated results will be saved into a Python dictionary DC and a file of out/DC.dat.

To plot the model results, ones can type the following code:

```
import pylab as plt
plt.loglog(DC['time'], DC['CO']/DC['nH']) # take CO as example
plt.show()
```

To combine GGCHEMPY with your models at any time step, please modify the integration part in the Python class ggchempy in "ggchempylib.py":

```
it = 0
while r.successful() and r.t < self.ggpars.tf * self.ggpars.yr_sec:
    ### to update rate coefficients:
    self.compute_reaction_rate_coefficients()
    r.integrate(self.ts[it])
    ...
    it+=1
```

where ts[it] is the current time with unit of second.

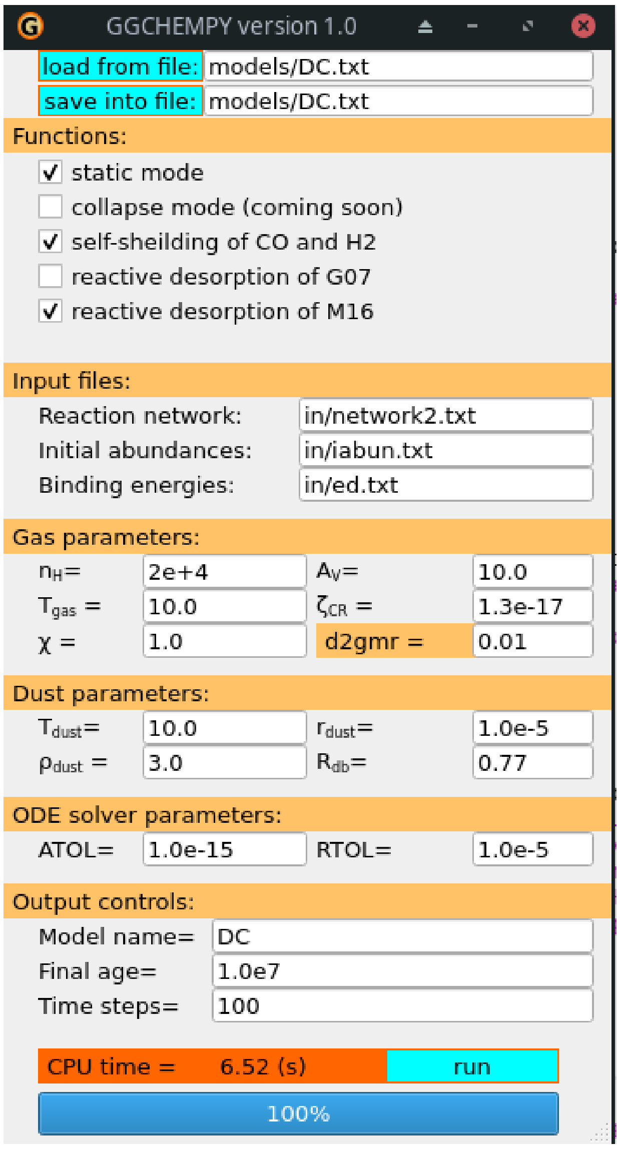
To add new formula to compute reaction rate coefficient, just modify function compute_reaction_rate_coefficients() according to your new reactions.

To load existing model and to format species for a figure, use the following functions:

```
from ggchempylib import loadgg, latex_species
```

- loadgg(): in case that ones want to load the previous result from GGCHEMPY with a file name of 'DC.dat' in folder of 'out', the function can be used as DC=loadgg('out/DC.dat').
- latex_species(): this function is used to produce chemical formula in latex format considering superscript and subscript. For example, latex_species("H3+") produces ${\rm H_3^+}$ which is compiled as H_3^+ in a figure.

2.4 Graphical user interface (GUI)



GGCHEMPY version 1.0

load from file: models/DC.txt

save into file: models/DC.txt

Functions:

- ✓ static mode
- collapse mode (coming soon)
- ✓ self-sheilding of CO and H2
- reactive desorption of G07
- ✓ reactive desorption of M16

Input files:

Reaction network:	in/network2.txt
Initial abundances:	in/iabun.txt
Binding energies:	in/ed.txt

Gas parameters:

n_H=	2e+4	A_V=	10.0
T_{gas} =	10.0	ζ_{CR} =	1.3e-17
χ =	1.0	d2gmr =	0.01

Dust parameters:

T_{dust}=	10.0	r_{dust}=	1.0e-5
ρ_{dust} =	3.0	R_{db}=	0.77

ODE solver parameters:

ATOL=	1.0e-15	RTOL=	1.0e-5

Output controls:

Model name=	DC
Final age=	1.0e7
Time steps=	100

CPU time = 6.52 (s)

Figure 2: Graphical user interface of GGCHEMPY.

GGCHEMPY also provides a GUI using PyQt5. To use GGCHEMPY via GUI, type the following code in Python script:

```
from ggchempylib import run_GUI
run_GUI()
```

A screenshot of GGCHEMPY GUI is shown in Fig. 2. Tips will appear when you hover mouse over a button or text label. By clicking `"load from file"` and `"save into file"`, ones can load previous model parameters storing in the following file and save current model parameters into the following file, respectively. To run a model, just click the `"run"` button. The CPU time used by the ODE solver will be printed when your model is finished.

2.5 Benchmark

To benchmark GGCHEMPY, the five static models from Semenov et al. (2010) are used: TMC1, HOTCORE, DISK1, DISK2 and DISK3. These models cover wide physical conditions of typical dark clouds, hot molecular core and three positions in a planetary disk which are good to verify the GGCHEMPY code validity by varying various parameters. The default parameters listed in Table 1 are used except for the varying parameters shown in the following code block. To ensure the same setting to that of Semenov et al. (2010), the diffusion-to-binding energy ratio is set to 0.77. The low-metal initial abundances are used. No reactive desorption is used (`iswitch.iNTD=0`). All the models are with

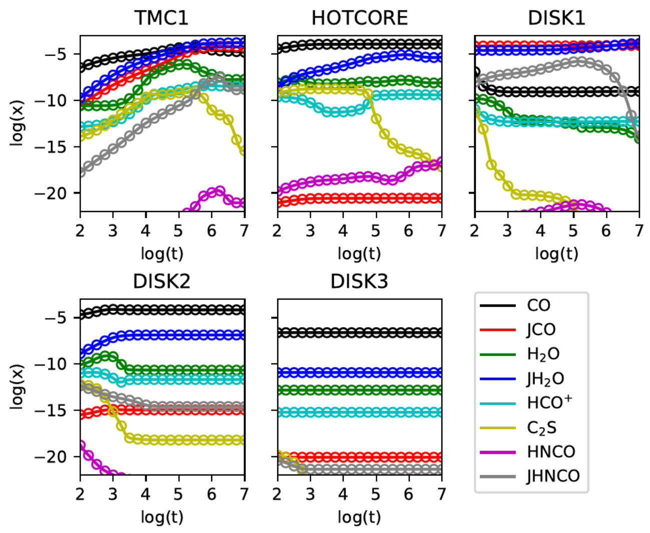


Figure 3: Comparison of selected species in the five models of TMC1, HOTCORE, DISK1, DISK2 and DISK3. Line and point with the same color indicate the same species from GGCHEMPY and ALCHEMIC codes (Semenov et al. 2010), respectively.

ggpars.rtol=1e-5 (relative tolerance) and ggpars.atol=1e-15 (absolute tolerance). A laptop with Linux system and Intel i7-3612QM (4 cores, 2.1 GHz) is used. The Python version is 3.8.5.

```
from ggchempylib import ggchempy as gg
### set input files:
gg.ggpars.ggfiles = \
['in/network.txt','in/ed.txt','in/iabun.txt']
gg.dust.surface.Rdb=0.77
modelnames = ['TMC1','HOTCORE','DISK1','DISK2','DISK3']
pars={}         # nH,     Tgas,   Av,   Chi, Tdust
pars['TMC1']   = [2.00e4, 10.0, 10.0, 1.0, 10.0]
pars['HOTCORE']= [2.00e7, 100.0, 10.0, 1.0, 100.0]
pars['DISK1']  = [5.41e8, 11.4, 37.1, 428.3, 11.4]
pars['DISK2']  = [2.59e7, 45.9, 1.94, 393.2, 45.9]
pars['DISK3']  = [3.67e6, 55.2, 0.22, 353.5, 55.2]
for model in modelnames:
    gg.gas.nH, gg.gas.T, gas.Av, gg.gas.chi, \
    gg.dust.T = pars[model]
    modelx = gg.run(model)
```

In Fig. 3, we show results of selected gas-phase and surface species (with prefix 'J') including neutral, ionic, C-, N-, O- and S-bearing species. From this figure, we see that GGCHEMPY (solid lines) reaches prefect agreements to the models of Semenov et al. (2010) (points).

The CPU times used by the five models are shown in Fig. 4. To reach a final age of 10^9 yr, the CPU times used by the GGCHEMPY in Python without Numba is about $\sim 200-300$ s for each model (orange). The CPU time used by GGCHEMPY using Numba is reduced to $\sim 10-15$ s (green). As a summary, GGCHEMPY using Numba reaches a comparable speed to the Fortran version GGCHEM (blue).

2.6 Updates to reaction network

Using the original reaction network of Semenov et al. (2010), we found that the modeled abundance of HNCO is $\sim 10^{-18}$ at 10^6 yr in the TMC1 model (purple color in Fig. 3) which is due to that some gas-phase reactions are missed. After adding the missed gas-phase reactions listed in Table 2, the modeled HNCO abundance reaches about 10^{-10} around 10^6 yr (line with magenta color) which is now comparable to the observed one (4×10^{-10}) (Agúndez & Wakelam 2013). We save the updated reaction into in/network2.txt. For studies focusing on HNCO, the updated

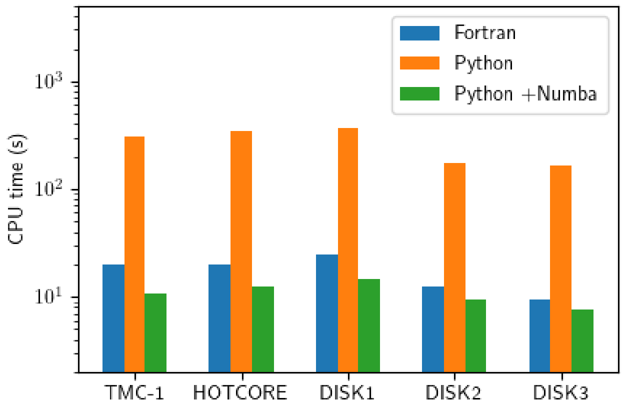


Figure 4: CPU times of the five benchmark models using the GGCHEM in Fortran (blue), Python (orange), Python+Numba (green), on the same laptop computer.

Table 2: Added gas-phase reactions.

Reaction	α	β	γ
$CH_2 + NO \longrightarrow HNCO + H$	3.65×10^{-12}	0	0
$H_2NCO^+ + e^- \longrightarrow HNCO + H$	1.00×10^{-7}	-0.5	0
$H_2NCO^+ + e^- \longrightarrow NH_2 + CO$	1.00×10^{-7}	-0.5	0
$H_3{}^+ + HNCO \longrightarrow H_2NCO^+ + H_2$	3.69×10^{-9}	-0.5	0
$HNCO^+ + H_2 \longrightarrow H_2NCO^+ + H$	1.51×10^{-9}	0	0
$HNCOH^+ + e^- \longrightarrow HNCO + H$	1.00×10^{-7}	-0.5	0
$HNCOH^+ + e^- \longrightarrow NH + HCO$	1.00×10^{-7}	-0.5	0
$H_3{}^+ + HNCO \longrightarrow HNCOH^+ + H_2$	3.69×10^{-9}	-0.5	0
$HNCO^+ + H_2 \longrightarrow HNCOH^+ + H$	1.51×10^{-9}	0	0
$He^+ + HNCO \longrightarrow HNCO^+ + He$	5.68×10^{-9}	-0.5	0
$NCO^+ + H_2 \longrightarrow HNCO^+ + H$	1.51×10^{-9}	0	0
$OCN + H_3{}^+ \longrightarrow HNCO^+ + H_2$	1.64×10^{-8}	-0.5	0
$HNCO^+ + e^- \longrightarrow OCN + H$	1.50×10^{-7}	-0.5	0

network should be used.

References

Agúndez, M. & Wakelam, V. (2013), 'Chemistry of Dark Clouds: Databases, Networks, and Models', *Chemical Reviews* **113**, 8710–8737.

Draine, B. T. (1978), 'Photoelectric heating of interstellar gas.', *ApJS* **36**, 595–619.

Garrod, R. T., Wakelam, V. & Herbst, E. (2007), 'Non-thermal desorption from interstellar dust grains via exothermic surface reactions', *A&A* **467**, 1103–1115.

Ge, J., Mardones, D., Inostroza, N. & Peng, Y. (2020), 'The roles of polycyclic aromatic hydrocarbons in dark cloud chemistry: new constraints on sulphur-bearing species', *MNRAS* **497**(3), 3306–3322.

Ge, J. X., He, J. H. & Li, A. (2016), 'Interstellar chemical differentiation across grain sizes', *MNRAS* **460**, L50–L54.

Ge, J. X., He, J. H. & Yan, H. R. (2016), 'Effects of turbulent dust grain motion to interstellar chemistry', *MNRAS* **455**, 3570–3587.

Ge, J. X., Mardones, D., He, J. H., Rawlings, J. M. C., Liu, S.-Y., Lee, J.-E., Tatematsu, K., Liu, T., Zhu, L., Chang, Q., Inostroza, N. & Feng, S. (2020), 'Three-dimensional Projection Effects on Chemistry in a Planck Galactic Cold Clump', *ApJ* **891**(1), 36.

Minissale, M., Dulieu, F., Cazaux, S. & Hocuk, S. (2016), 'Dust as interstellar catalyst. I. Quantifying the chemical desorption process', *A&A* **585**, A24.

Momcheva, I. & Tollerud, E. (2015), ‘Software Use in Astronomy: an Informal Survey’, *arXiv e-prints* p. arXiv:1507.03989.

NSF (2017), ‘’, *Committee on Software Infrastructure for Heterogeneous Computing, https://github.com/labarba/NSFcommittee-SI2017/* .

Semenov, D., Hersant, F., Wakelam, V., Dutrey, A., Chapillon, E., Guilloteau, S., Henning, T., Launhardt, R., Piétu, V. & Schreyer, K. (2010), ‘Chemistry in disks. IV. Benchmarking gas-grain chemical models with surface reactions’, *A&A* **522**, A42.

Sipilä, O., Caselli, P., Redaelli, E., Juvela, M. & Bizzocchi, L. (2019), ‘Why does ammonia not freeze out in the centre of pre-stellar cores?’, *MNRAS* **487**(1), 1269–1282.

Tafalla, M., Myers, P. C., Caselli, P. & Walmsley, C. M. (2004), ‘On The Internal Structure Of Starless Cores. Physical and Chemical Properties of L1498 and L1517B’, *APSS* **292**(1), 347–354.

Tafalla, M., Santiago-García, J., Myers, P. C., Caselli, P., Walmsley, C. M. & Crapsi, A. (2006), ‘On the internal structure of starless cores. II. A molecular survey of L1498 and L1517B’, *A&A* **455**(2), 577–593.

Tang, M., Ge, J. X., Qin, S.-L., Liu, T., Wu, Y., Kim, K.-T., Liu, S.-Y., Zhang, C., He, J. H., Ju, B.-G. & Fang, X. (2019), ‘Chemical Properties of Two Dense Cores in a Planck Galactic Cold Clump G168.72-15.48’, *ApJ* **887**(2), 243.

Tritsis, A., Yorke, H. & Tassis, K. (2018), ‘Python Radiative Transfer Emission code (PYRATE): non-local thermodynamic equilibrium spectral lines simulations’, *MNRAS* **478**(2), 2056–2064.